**Python Program for Time and Day of any Place (PythonDayTimeProject)**

**Objective:**

The goal of the project is to create a Python program that shows the current time and day of the week for any location around the world. The user can type the name of a place, and the program will fetch the time and day based on the location’s time zone. It’s made with a simple UI and a simple double-click will run the program.

This project consists of a batch file and a Python script that shows the current day and time of a searched place. To run the program, simply double-click the "DayTime.bat" file, ensuring the correct path to the Python script. The batch file runs the "DayTimePython.pyw" Python script without opening a command prompt window, providing a clean and simple user experience.

## How the program works from start to finish:

Project name: PythonDayTimeProject.

A batch file Python program that shows the day and time of a searched place.

**How to Run:**

Double-click "DayTime.bat" to run the program, ensuring the correct path to the Python file.

The repository contains two files:

1. “DayTime.bat”: The batch file that runs the "DayTimePython.pyw" Python script.

2. “DayTimePython.pyw”: The Python script for the program.

**“DayTime.bat” Breakdown:** This Windows batch file runs the Python script.

**Here’s what each line does:**

1. ‘@echo off’: Hides commands from the Command Prompt for a cleaner look.

2. ‘start pythonw C:\Users\Admin\Desktop\DayTimePython.pyw’:

- ‘start’ runs the Python script in a new window.

- ‘pythonw’ runs the script without opening the Command Prompt window.

- ‘C:\Users\Admin\Desktop\DayTimePython.pyw’ is the path to the Python script.

3. ‘exit’: Closes the batch file after execution.

**Commands:**

- pythonw: A Windows interpreter that runs Python scripts without showing the console.

- .pyw: A Python script file extension that runs without a console window when executed.

The batch file ensures the script runs without showing the console, even if ‘.pyw’ isn't properly associated with ‘pythonw.exe’.

**Libraries Used in Python program “DayTimePython.pyw”:**

In this project, I used both **built-in** and **external** Python libraries to handle the tasks of creating a GUI, fetching location data, and working with time zones. Here’s a breakdown of each library:

1. **Tkinter**:
   * Tkinter is a built-in Python library used to create Graphical User Interfaces (GUIs).
   * It allows to create windows, buttons, labels, and other interactive elements.
   * In the program, Tkinter is used to display a window where users can enter a place name and view the time and day for that place.
2. **Datetime**:
   * The datetime library helps us handle dates and times.
   * It allows to get the current date and time, as well as format it in a readable way.
   * In this program, I used datetime.now() to get the current time, and strftime() to format it as a string (e.g., "12:30 PM").
3. **Pytz**:
   * pytz is an external library that helps us work with time zones.
   * It allows us to convert time to different time zones and make sure it's accurate.
   * In this program, I used pytz.timezone() to get the time for a specific location’s timezone.
4. **Geopy**:
   * geopy is an external library used for geocoding, which means converting a place name (e.g., "New York") into geographic coordinates (latitude and longitude).
   * In the program, I used the geolocator.geocode() function to get the coordinates of a place, which are needed to find its timezone.
5. **TimezoneFinder**:
   * timezonefinder is an external library used to find the timezone based on geographic coordinates (latitude and longitude).
   * I used timezone\_finder.timezone\_at() to determine the timezone for the given place’s coordinates.

**How the Program Works in “DayTimePython.pyw”:**

1. User Input

* The user enters the name of a place (e.g., "New York") in a text input field in the Tkinter window.

2. Geocoding: Converting Place to Coordinates

* The program uses Geopy’s Nominatim geocoder to convert the entered place name into latitude and longitude (the geographic coordinates).
* Example: If the user enters "New York", the program will find the coordinates of New York (latitude: 40.7128, longitude: -74.0060).

3. Finding the Timezone

* Once the coordinates are obtained, the program uses TimezoneFinder to find the timezone for that location based on its latitude and longitude.
* Example: For New York, the timezone would be "America/New\_York".

4. Getting the Current Time

* The program uses Pytz to handle the time zone.
* It converts the local time in the found timezone to the correct time for the location.
* Example: If the user is in New York, the program will return the local time in New York (e.g., "12:30 PM").

5. Displaying the Time and Day

* The program then displays the current time (in 12-hour format with AM/PM) and the day of the week (e.g., "Monday") for the entered place in the Tkinter window.

**Code Structure:**

**GUI (Graphical User Interface)**

* I used Tkinter to create the window, buttons, labels, and text fields.
  + Tk() creates the main window where everything will be displayed.
  + Label() is used to show text on the window (e.g., instructions, time, and day).
  + Entry() creates a text box where the user can type the name of the place.
  + pack() arranges the widgets in the window.
  + bind() connects an action (like pressing the Enter key) to a function (get\_time\_and\_day()).

**Getting Data and Processing Time**

1. User Input: The user enters the place name in the Entry field and presses Enter.
2. Geocoding: The place name is passed to Geopy’s geocode() method to get latitude and longitude.
3. Timezone: The coordinates are passed to TimezoneFinder to get the timezone of the place.
4. Time Calculation: The timezone is passed to Pytz to get the local time, and I use datetime to format and display the time.
5. Display: The time and day are shown in the Tkinter window.

**Example:**

Let’s go through an example. The user enters "Paris" in the text box:

1. The program uses Geopy to get the coordinates for Paris.
2. It then uses TimezoneFinder to determine the timezone for those coordinates (e.g., "Europe/Paris").
3. Using Pytz, it fetches the current time in Paris and formats it (e.g., "10:30 AM").
4. Finally, it displays the time and day (e.g., "Monday") in the window.

## Link to Tutorials/Materials Used:

1. Python Documentation: For understanding how the Tkinter, datetime, and other built-in libraries work.
   * [Python.org Documentation](https://docs.python.org/3/)
2. Geopy Tutorial: For learning how to use Geopy to fetch location details like latitude and longitude.
   * [Geopy Documentation](https://geopy.readthedocs.io/)
3. TimezoneFinder Documentation: To understand how to find the timezone of a given location using latitude and longitude.
   * [TimezoneFinder on GitHub](https://github.com/jannikmi/timezonefinder)
4. Pytz Documentation: For converting time between different timezones.
   * [Pytz Documentation](https://pythonhosted.org/pytz/)
5. YouTube: Had to watch videos on how to use the libraries required.
6. Stackoverflow: How to use libraries and error handling.
7. Batch File Basics: For automating the execution of the Python script with a batch file.
   * Various online tutorials on batch file scripting, such as those on [GeeksforGeeks](https://www.geeksforgeeks.org/).

## Customizations Made:

1. Wrote and modified code:
   * Wrote some and edited most of the code to match with my required UI, functionality and outputs.
   * Most of the help, I got was from the library’s documentation/Stackoverflow and looking at how they used the library on YouTube.
2. Simple GUI with Tkinter:
   * Created a user-friendly interface with input fields, labels, and a clean layout.
   * Added error messages using message boxes to guide the user if something goes wrong.
3. Batch File for Easy Execution:
   * Added a DayTime.bat file to allow users to run the program with just a double-click.
   * Used pythonw in the batch file to ensure the program runs without a console window popping up.
4. Error Handling:
   * If the user enters an invalid place name, the program shows a clear error message.
   * Handles cases where the location or timezone can’t be found gracefully.
5. No-Console Execution:
   * Ensured the program runs in the background with no command prompt showing, making it look more professional.
6. Improved Code Structure:
   * The code is organized into functions for better readability and maintainability.

## What I Learned:

1. Tkinter for GUI Development:
   * Learned how to create windows, input fields, buttons, and labels for a graphical interface.
   * Used message boxes to display error messages or alerts to the user.
2. Geopy Library:
   * Understood how to use Geopy to fetch location details like latitude and longitude based on a place name.
3. TimezoneFinder Library:
   * Learned to find the timezone of a location using latitude and longitude.
   * Understood how to handle cases where timezone data isn’t available.
4. Pytz Library:
   * Used Pytz to convert and display the current time in a specific timezone.
   * Explored how to format time and dates for better readability.
5. Datetime Module:
   * Learned how to work with the datetime library to get the current time and day.
   * Explored how to format time into 12-hour format with AM/PM.
6. Batch Files:
   * Learned how to automate program execution using a .bat file.
   * Understood the purpose of commands like @echo off, start, and exit.
7. Error Handling:
   * Improved my skills in debugging and handling exceptions gracefully in Python.
8. Combining Tools and Libraries:
   * Discovered how to integrate multiple libraries and tools to create a seamless and functional application.
9. User Experience Enhancements:
   * Focused on making the program simple and user-friendly, ensuring it could be run easily without technical knowledge.

**Conclusion:** By using:

* Tkinter for the GUI,
* Geopy for geocoding,
* TimezoneFinder for timezone lookups,
* Pytz for time zone management,
* Datetime for time handling,

I was able to create a program that can tell the current time and day of any place in the world.

I also learned how to:

* Create a simple GUI for user input.
* Using external libraries like Geopy, TimezoneFinder, Pytz, and Datetime to fetch and manipulate data.
* How to read/understand official documentation of languages and libraries.
* Use geocoding to convert place names to coordinates.
* Handle time zones and local time using external libraries.
* How to handle user input, geocode it, and display results interactively.
* Display the results in a simple user-interactive window.

This project helped demonstrate the power of Python libraries and how they can be used together to build useful applications.